Commitment Scheme For Bitcoin

Muhammad Naufal Ashshiddiq Wangsaatmadja
Student ID: 1597429
Supervisor: Dr. David Galindo

Submitted in conformity with the requirements
for the degree of MSc Cyber Security

School of Computer Science
University of Birmingham
September 2016

Abstract

Threshold ECDSA is a promising method to mitigate malware in Bitcoin Wallet. By sharing the signing power, it is reliable against a single point of failure. It prevents a malware attempting to steal the private key to obtain the Bitcoin and provide a backup mechanism for the wallet. It consists of a complicated cryptosystem.

A commitment scheme is one of the building blocks in the threshold ECDSA. Damgard, Gennaro, and MacKenzie commitment are listed as applicable commitment scheme for the threshold ECDSA. Studying the three commitment lead us to an assumption that none of them is efficient. Galindo scheme is proposed in return as an option.

By implementing the commitment schemes and analysing the computation time when committing a message, It is proven that the proposed scheme is more efficient in average by 50%.

Keywords: Bitcoin, Threshold ECDSA, Commitment scheme, Efficient

Acknowledgements

I would like to express my gratitude to The Almighty, Allah, because of his blessings that I can finally complete this project that is named "Commitment Scheme For Bitcoin". I know that finishing this project will not be possible without help and support from the other peoples. So, I would like to say thanks to.

- My supervisor Dr David Galindo for his tremendous patience, guidance, and assistance during this project. It would not have been possible to finish the project without his constant support.
- Mark Ryan and Paul Levy for their feedback for this project.
- My family and friends for the constant support for the completion of this project.
- Rifa Aprilianti for her patience, love, and support.
- Bancha, Karishma, Nurul, Mas Amir, and all of the fellows from the Msc. Cyber Security programme 2015/2016. It's been a fantastic year guys!
- All of the staff PPI-Birmingham for being considerate and always supporting me finishing this project.
- All of the people that I cannot mention one by one that has been supporting me completing this project.
- Indonesia Endowment Fund for Education (LPDP -Lembaga Pengelola Dana Pendidikan), Ministry of Finance, The Republic of Indonesia, for their generous scholarship all the way to complete the degree of MSc Cyber Security

Hopefully, all the people that have been supporting and helping me get the blessing of God and this dissertation is useful for the one that needed.

Contents

Abstract ii

Acknowledgements iii

1 Introduction 1
- 1.1 Motivation . . . 2
- 1.2 Contribution of the project . . . 2
- 1.3 Related work . . . 2
- 1.4 Project Outline . . . 3

2 Further background material 4
- 2.1 Bitcoins . . . 4
- 2.2 Bitcoin Wallet . . . 4
- 2.3 Secret Sharing . . . 5
- 2.4 Threshold DSA/ECDSA . . . 6
- 2.5 Independent Trapdoor Commitments . . . 8
- 2.6 Damgard Commitment . . . 9
- 2.7 Gennaro Commitment . . . 10
- 2.8 MacKenzie Commitment . . . 11
- 2.9 Galindo Commitment . . . 11

3 Analysis and Specification 13
- 3.1 Galindo Independent Trapdoor Commitment based on GGRO8 . . . 13

4 Design 16
- 4.1 Input and Output . . . 16
- 4.2 SageMath programs . . . 17

5 Implementation and Testing 18
- 5.1 Implementation . . . 18
- 5.2 SageMath Program . . . 19

5.3 Testing Design 25

6 Results and evaluation 26

6.1 SageMath program results 26

6.2 Testing Result 28

7 Discussion 31

7.1 Achievements 31

7.2 Future Work 31

8 Conclusion 32

Appendix 35

A SVN project repository 35

A.1 Contents of the SVN project repository 35

A.2 How to run our software 35

Chapter 1

Introduction

Bitcoin is a cryptocurrency protected by the elliptic curve digital signature [1]. To sign a transaction (spend the Bitcoin), individuals needs to use a private key. If a malware / adversary can get the private key, they will be able to steal the Bitcoin. That is why the security of the private key is vital in the Bitcoin system. Threshold ECDSA in [1] is an alternative to mitigate the malware stealing Bitcoin.

Threshold ECDSA (k, n) enables a group of n to share a signing power where each in the group cannot sign without collectively signing with at least k share. Consequently, an attacker needs to have at least k share to be able to sign a transaction. This increase the security of the Bitcoin. Because $k < n$, some *share* can also be used as an backup. Threshold ECDSA solves the single point of failure that currently resides in the single Bitcoin signature. Threshold ECDSA consists of complex protocol.

One of the protocol is the commitment scheme. There are three listed candidates commitment scheme that are applicable for the threshold ECDSA which is constructed by Damgard [2], Gennaro [3], and MacKenzie [4]. All of those has different ways to implement the key generation and to commit a message, but they have something in common which is a large number of exponent computation. The large number of exponentiation lead those three schemes into an inefficient scheme.

The Author with the help of supervisor proposed a new commitment based on [5] that is also applicable for the threshold ECDSA but more efficient. The project will analyse the efficiency of the proposed scheme by implementing a program that can run the key generation and commit algorithm for all of the stated schemes before and then comparing the efficiency of them. The efficiency will be measured by the processing time for the scheme to commit a message.

[1] https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm

1.1 Motivation

The motivation of this project is as follows:

- Learning how to mitigate malware stealing Bitcoins using Threshold ECDSA
- Learning the Damgard, Gennaro, MacKenzie commitment scheme
- Proposing a more efficient commitment scheme
- Implementing the Damgard, Gennaro, MacKenzie, and the proposed commitment scheme
- Comparing the computation time as a proof of efficiency

1.2 Contribution of the project

The main contributions of the project are proposing a new commitment scheme that is efficient and applicable for the threshold ECDSA which also better than the other commitment scheme listed in [1]. Analysis and implementation of each scheme in SageMath program are used as a proof of concept.

1.3 Related work

Researchers have been working on about the Threshold DSA/ECDSA and the commitment scheme.

The paper "Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security" posted in 2016 [1] is a start for this project. It explains the importance and benefit of implementing threshold ECDSA in Bitcoin Wallet, explains the complex building block for the cryptosystem including the option of the commitment scheme. Authors of the paper have developed the prototype of the threshold ECDSA for the Bitcoin wallet and share it as an open source.

The paper "Non-interactive and reusable non-malleable commitment schemes" posted in 2003 [2], "Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks" posted in 2004 [3], and "On simulation-sound trapdoor commitments" posted in 2004 [4] elaborates about the construction of their commitment schemes. Those papers provide the mathematical detail which are used as a foundation to realise the schemes.

The paper "Computational soundness of non-malleable commitments" posted in 2008 [5] provides a generic construction for a non-malleable commitment scheme

based on one-way trapdoor permutations. This paper is proposed as an option for a more efficient commitment that is also applicable for the threshold DSA/ECDSA. To realise the tag based encryption, the paper "The twin Diffie-Hellman problem and applications" posted in 2008 [6] which provided a variant of the Cramer-Shoup scheme is chosen. Furthermore, the paper "Efficient hybrid encryption from ID-based encryption" posted in 2010 [7] containing CMA-Secure one-time signature based on DLP which in this project is used to realise the signature scheme.

In conclusion, this project is significantly related to the above papers since every part of our project aims to realise mathematical details of the commitment scheme.

1.4 Project Outline

The systematical approach for this report is as follows:

1. Chapter 2 Further Background Material
   This section consists of the basic theory of Bitcoins, Bitcoin Wallet,Threshold ECDSA, Independent Trapdoor Commitments,Damgard Commitment, Gennaro Commitment, MacKenzie Commitment, Galindo Commitment

2. Chapter 3 Analysis
   This section consists of the analysis on Damgard Commitment and realisation of the proposed Galindo commitments.

3. Chapter 4 Design
   This section consists of the design of the input output specification and the design implementation of each scheme in SageMath.

4. Section 5 Implementation and Testing
   This section consists of implementation code and the testing design.

5. Chapter 6 Result and Evaluation
   This section consists of unit testing of the SageMath implementation for each scheme, result of the testing designed in Chapter 5, and the evaluation of the result.

6. Chapter 7 Discussion
   This section consists of the Achievement the future work.

7. Chapter 8 Conclusion This section consists of the conclusion of the project.

Chapter 2

Further background material

In this chapter, the background materials that are used as a foundation of this project will be described.

2.1 Bitcoins

Bitcoins is a cryptocurrency, an electronic payment system based on cryptography. Bitcoin uses peer-to-peer technology to operate with no central authority or banks; managing transactions and the issuing of bitcoins is carried out collectively by the network [1]. It shares a public ledger called a block chain. To send bitcoin, user will need to sign a transaction using its private key and then send it to the network. Other users can then verify the transaction using the public key. After the transaction is signed by the user and send to the network, miners will try to confirm the waiting transactions to finally include it in the block chain as a valid transaction by solving a mathematical problems with a rewards that is specified by the users.
The private key is the only proof of the ownership of the address which can be used to spend the bitcoin. Losing the private keys means losing access to the money. By having the access to the private keys, adversary can impersonate the real user and steal the key.

2.2 Bitcoin Wallet

Bitcoin wallet is an abstraction of a physical wallet which main purpose is to to hold set of private keys with the corresponding public keys [8]. Backing up the keys become essentials because it is the only proof of the address ownership and

[1] https://bitcoin.org/en/

if it losed, it could not be regained. There are several types of bitcoin wallet such follows:

- Web / Online Wallets
  The private keys are stored online using computer controlled by the server provider. It is convenient because it is accessable anywhere but the main disadvantage is that rellying on the other security. If the online system is breached, we have a chance to lose our coins. Moreover there is a chance that the provider steals our money as well because they store the keys. Example of this wallet are CoinBase, BlockChain.

- Desktop Wallet
  Wallet is installed in a PC that can be used to create bitcoin address to send and receive bitcoins. This wallet is great if the user are able to secure their PC from malware. Example of this wallet are BitcoinCore, MultiBit, BitcoinKnots.

- Mobile Wallets
  Wallet is installed in mobile devices. The example of this wallet is Bitcoin Wallets.

- Hardware Wallets
  This wallet uses a specially design hardware to store the keys which is secure against malware. The example of this wallet are Trezor, Ledger, KeepKey.

- Paper Wallets
  The private and public keys of the wallet are printed into a paper. It provides high security because it prevents any attacker to stole the keys using malware. It needs physical access to hold the papers. The disadvantage of this wallet is the quality of the ink which may degradate through time or in some cases the user lost the paper.

2.3 Secret Sharing

Encrypting a message with a key has a certain weakness. It depends on the keys. If the key is lost, the message will not be able to be reconstructed. Moreover, if the key is possessed by the adversary, the secrecy of the message will be lost. As a result, protecting the secrecy of the key is become the focus of securing message. Encrypting a key with another key is insecure where on the other hand replicating key will not solve the problem. [9].

Secret sharing (k, n) is a method to conceal a message by distributing the secret message into the group of a n participant in a shape of share [10]. The message can only be reconstructed if a k number of share are combined. $k-1$ number of *share* will not leak any information. Secret sharing disables the ability of the owner of the shared secret to reconstructed the message independently which resulted in enhancing the security of the message if it is compared to the other encrypting method that is using single key.

2.4 Threshold DSA/ECDSA

Threshold ECDSA adopts the nature of the secret sharing. The private key which is used to sign a message can be shared in a form of share to n participants which only when k participant do the distributed signing they can sign the key. The application for this scheme is that people can make a share of their key, and then they try to put it in different place or different device. In Bitcoin environment for example in the $(2, 2)$ scheme, people can create the share of the key, and put one part of the key in phone and the other part in the computer. It prevents the malware that might attempts to steal the key. It needs to infect both device to be able to compromise the key. The other function that might be used is to create more share as a backup. Remember that if the key is lost, it means that there is no way to have the access for the Bitcoin. So sharing the key is a good way to not only secure it but also makes it reliable.
The scheme proposed in [1] consist of a complex protocol. There are building block that needs to be done as follows.

1. Initialization phase
   In this phase, common parameters G,g,q for the DSA are choosen. If its build on top of the Bitcoin environment, the value of G is on a shape of coordinate.

2. Key generation
   For the players to jointly generate a DSA key pair $(x, y = g^x)$, the idea is to generate a public key E and secret key D in shared form among the player for additively homomorphic encryption (Mod N). The N is chosen to be larger than q^8. Then a value x is generated and encrypted with E in this case is a paillier cipher with value $\alpha = E(x)$ made public [1]. To enforce the independence values contributed by each player to the selection of x, the scheme use independent trapdoor commitments. Each player computers a ZK argument $\prod_i$. For each round, if the commitments open to null or any ZK proof fails,

the protocol terminates without an output.

3. Signature Generation
   For the player to jointly generate a signature, there are six rounds that must be followed.

   (a) Round 1
   For each player P_i

   - select ρ_i randomly from Z_q
   - compute $u_i = E(\rho_i)$
   - compute $v_i = E(\rho_i x)$
   - Commit the u_i and v_i
     $[C_{1,i}, D_{1,i}] = Com([u_i, v_i])$
     then Broadcast the Commitment $C_{1,i}$

   (b) Round 2
   Each player P_i broadcast the Decommitment of $C_{1,i}$, $D_{1,i}$ and the Zero knowledge argument of $\prod_{1,i}$. Furthermore, players do some computation as follows:

   - Calculate $u = E(\rho)$
     where $\rho = \sum_{i=1}^{t+1} \rho_i$
   - Calculate $v = E(\rho x)$

   (c) Round 3
   For each player P_i

   - select k_i randomly from Z_q
   - select c_i randomly from $[-q^6, q^6]$
   - compute $r_i = g^{k_i}$
   - compute $w_i = E(k_i \rho_i + c_i q)$
   - Commit the r_i and w_i
     $[C_{2,i}, D_{2,i}] = Com([r_i, w_i])$
     then Broadcast the Commitment $C_{2,i}$

   (d) Round 4
   Each player P_i broadcast the Decommitment of $C_{2,i}$, $D_{2,i}$ and the Zero knowledge argument of $\prod_{2,i}$. Furthermore, players do some computation as follows:

   - Calculate $k = \sum_{i=1}^{t+1} k_i$

- Calculate where $c = \sum_{i=1}^{t+1} c_i$
- Calculate $R = g^k$
- Calculate $r = H'(R)$ in Z_q

(e) Round 5
Each Player computes $\sigma = E(k^{-1}(m + xr)) = E(s)$

(f) Round 6
Players invoke distributed decryption protocol TDec over ciphertext σ. Let $s = D(\sigma) mod q$. Players then output (r, s) as a signature for m.

2.5 Independent Trapdoor Commitments

A trapdoor commitment scheme allows a sender to commit to a message with information-theoritic privacy i.e., given the transcript of the commitment message the receiver even with infinitie computing power, cannot guess the committed messages better than at random [1].
Formally a non-interactive trapdoor commitment scheme consist of four algorithm KG, Com, Ver, Equiv with the following properties :

- KG (Key Generation)
  Input : Security parameter
  Output : Pair of pk (public key associated with the commitment scheme) and tk (trapdoor)

- Com (Commitment Algorithm)
  Input : pk and message M
  Output : $[C(M), D(M)] = Com(pk, M, r)$ with r are coin tosses, $C(M)$ is commitment string, $D(M)$ is decommitment string that is kept secret until opening phase

- Ver (Verification Algorithm)
  Input : C, D, pk
  Output : messages M or null

- Equiv (Open Commitment)
  Input : pk, M, r, message $M' \neq M$, string T
  Output : if $T = tk$, returns D' such that $\text{Ver}(pk, C(M), D') = M'$

In [1], there are three commitment scheme that is applicable for building the crypto block of the threshold DSA. Those three are Damgard Groth commitment [2],

Gennaro RSA commitment [3], MacKenzie commitment [4]. By analysing those list commitment algorithm, it can be found that those are not efficient. By the help of my supervisor, we propose the fourth commitment scheme which are based on [5]. All of those four commitment will be described in the next section.

2.6 Damgard Commitment

Damgard Groth proposed a construction of non-malleable commitment schemes that implementation is based on the strong RSA assumption in [2]. The scheme consist of phases as follows:

- KG
  This phase is used to generate the public key pair (N, s, e) and the trapdoor (p,q). The detail of the phase is as follows:

  1. Select n as a k-bit RSA modulus.
  2. Select q as 2k+1 bit prime.
  3. Select y randomly from Z_n^*.

- Com
  To commit to an element x $\in Z_q$, there are steps that needs to be followed:

  1. Let n an RSA modulus and y random number from Z_n*
  2. select r randomly from Z_n^*.

  3. Let w signature of α where $y = w^{\alpha} mod n$
  4. Define function h
     By using $c \in Z_n^*$, this function outputs smallest prime larger than $2^k c$.
  5. Message authentication scheme
     The authentication key is $ak = (r_1, r_2)$ with
     r_1 is picked at random from Z_n^*
     r_2 is picked at random from Z_n^*
     $mac = r_1 a + r_2 \ mod \ n$
  6. Relation $R = ((n, y, \alpha), w) | y = w^{\alpha} mod n$
     Prover sends a $= r^{\alpha}$
     Verifier sends random k-bit number m
     Prover sends $z = r w^m mod n$.

The Commitment C for k-bit message $= (c, a, mac)$
Decommitment $D = (m, d, z)$
where
$(c, d) = commit_{n,q,y}(r_1, r_2), (a, m, z) = S((n, y, \alpha), m)$

It could be observed that this commitment scheme is costly in computation. It is caused by the function h that is tempting to find the smallest prime larger than $2^k c$. It is bad because each time the scheme commiting a message, generating random prime number c and doing primality test is costly. So this scheme does not need to be investigated or implemented even further because in a concept it is already costly.

2.7 Gennaro Commitment

- KG
  To generate the public key pair (N, s, e) and the trapdoor (p, q), the process is as follows:

  1. Select p, q large primes
  2. Select s randomly from Z_N^*
  3. Compute $N = p * q$
  4. Compute $\phi = (p-1)(q-1)$
  5. Select e with $1 < e < q$ and $GCD(e, \phi(N)) = 1$

- Com
  To commit to a message $a \in [1..2^{l-1}]$, the steps are as follows:

  1. Select r randomly from Z_N^*
  2. Compute $A = s^a.r^e mod N$

  The commitment / decommitment pair (C, D) is

  $$(A, (a, r)) \tag{2.7.1}$$

- Ver
  To verify the commitment of the sender, sender reveals the a,r to the receiver and verify that $A = s^a.r^e mod N$

2.8 MacKenzie Commitment

SSCT scheme based on DSA consist of several steps as follows:

- TCGen
  This phase is used to generate DSA public and private key pair (Pk, Sk)
  $$Pk = (g, p, q, y) \qquad Sk = (g, p, q, x)$$
  value of each element has the specification as follows:
  g = generator
  p = a random prime
  q = prime order of a cyclic group that is generated by g
  x = random element in Z_q
  $y = g^x$

- TCcom
  To commit to a messsage m, it needs to select α and β randomly from Z_q. Then it needs to calculate g', h, c using Equation 2.8.2.
  $$\begin{aligned} g' &= g^{\alpha} mod p \\ h &= g^{H(tag)} y^{g'} mod p \\ c &= (g')^{\beta}.h^m \end{aligned} \tag{2.8.2}$$
  Resulted commitment/decommitment pair (C, D) can be seen in the Equation 2.8.3
  $$((g', c), \beta) \tag{2.8.3}$$

- TCver
  To verify the commitment, verify that
  $$c \equiv g'^{\beta} h^m \tag{2.8.4}$$
  with
  $$h \equiv g^{H(tag)} y^{g'} \tag{2.8.5}$$

2.9 Galindo Commitment

This scheme is proposed by looking at the other scheme that have larger number of exponentiation computation. In this scheme, let Tag based encryption scheme $\prod$ = (KeyGen,Enc,Dec) and Signature scheme $\sum$ = (Gen,Sign,Vrfy). The key generation, commit and verify algorithm of this scheme are as follows [5]:

- Key Generation
  To generate a key pair (pk, sk), the trusted third party (TTP) runs the KeyGen in tag based encryption twice.

$$\begin{aligned}(pk_1, sk_1) &= KeyGen(1^\eta)\\ (pk_2, sk_2) &= KeyGen(1^\eta)\end{aligned} \tag{2.9.6}$$

- Commit
  To commit into message m, it needs to select r_1,r_2 uniformly from Random R with R has the same message space as the message. Then, it calculates the pair of (vk, sk) using the Gen algorithm in the signature scheme.

$$(vk, sk) = Gen(1^\eta) \tag{2.9.7}$$

  Furthermore, it needs to calculate Ciphertext c_1 and c_2 using Enc algorithm on the tag based encryption.

$$\begin{aligned}c_1 &= Enc(pk_1, vk, m; r1)\\ c_2 &= Enc(pk_2, vk, m; r2)\end{aligned} \tag{2.9.8}$$

  Lastly, it calculates s which is the signature of the c_1 and c_2 using sk.

$$s = sign(sk, (c_1, c_2)) \tag{2.9.9}$$

  The resulted notation of commitment/decommitment (C, D) can be seen in the Equation 2.9.10

$$(C, D) = ((vk, c_1, c_2, s), (m, r_1)). \tag{2.9.10}$$

- Verify
  To verify the commitment, the receiver needs to check whether signature on $(c1, c2)$ is true and if $c_1 = \text{Enc}(pk_1, vk, m; r_1)$.

The realisation of this scheme that is applicable for commiting value for the pailier cipher (2047 for the exponent) is going to be described more in the third section.

Chapter 3

Analysis and Specification

In this chapter, the specifications of Galindo commitment that is applicable with the threshold DSA used in this project will be shown.

3.1 Galindo Independent Trapdoor Commitment based on GGRO8

Basic security parameter of the paillier cypher to be secured is the $|N| = 2047$ for the exponent. Encryption of the paillier cipher works on the mod of N^2. So to commit two value, (u_i, v_i) paillier cipher, which both are in $N^2 = 4094$, in total it needs minimum of 8188 bits length.

In this scheme, let Tag based encryption scheme $\prod$ = (KeyGen,Enc,Dec) as a Cramer-Shoup scheme [6]. Then let Signature scheme $\sum$ = (Gen,Sign) as an One-time Signature based on DLP [7]. The One-time signature consist of two algorithm as follows:

- Key Gen
  To generate a key pair private key $sk = (a, b, c)$ and public key $vk = (g_1, g_2, y)$ where
  $(a, b, c) \in Z_q 224$
  $(g_1 = g^a, g_2 = g^b, y = g^s)$

- Sign
  To calculates Signature σ of the c_1 and c_2 as follows.

$$\sigma = (\sigma_1, \sigma_2) \tag{3.1.1}$$

  where
  $\sigma_1 = q_{224}$

$$\sigma_2 = \frac{(c - m - a.\sigma_1)}{b} \text{ with } m = sha512(c_1, c_2)$$

So, The key generation, commit and verify algorithm of this proposed scheme are as follows [5]:

- Key Generation
  To generate a key pair (pk, sk), the trusted third party (TTP) runs the KeyGen GGR08.

$$\begin{aligned} (t_1, t_2) &\leftarrow Z_{q8192} \\ h_1 &= s_{8192}^t t_1 \\ h_2 &= h_1^{t_2} \end{aligned} \tag{3.1.2}$$

  Furthermore it runs the One-time signature keygen
  Then it runs the Cramer-Shoup KeyGen

$$\begin{aligned} (x_1, \tilde{x_1}, x2, \tilde{x_2}) &\leftarrow Z_{q8192} \\ (\tilde{X_1}, \tilde{X_2})(t_1, t_2) &\leftarrow Z_{q8192} \\ h_1 &= s_{8192}^t t_1 \\ h_2 &= h_1^{t_2} \end{aligned} \tag{3.1.3}$$

  where

$$\begin{aligned} X_i &= g^{x_i} \\ \tilde{X_i} &= g^{\tilde{x_i}} \end{aligned} \tag{3.1.4}$$

- Commit(pk,m,r)
  To commit into message $m \in Z_{q8191}$, it selects r randomly from Z_{q8191} then calculates c_1 and c_2.

$$\begin{aligned} c_1 &= h_1^m . h_2^{sha256(vk).r} mod p_{8192} \\ c_2 &= csEnc((x_1, \tilde{x_1}, x2, \tilde{x_2}), vk; m) \end{aligned} \tag{3.1.5}$$

  where
  $csEnc((x_1, \tilde{x_1}, x_2, \tilde{x_2}), vk; m)$ means calculating these values.

$$\begin{aligned} y &\in Z_q \\ t &= T(vk, Y) \\ Z_1 &= (X_1^t . \tilde{X_1})^y \\ Z_2 &= (X_2^t . \tilde{X_2})^y \\ k &= sha256(X_1^y) \\ c &= AES256_k(m) \end{aligned} \tag{3.1.6}$$

Output of the csEnc is a ciphertext (Y, Z_1, Z_2, c).
$\sigma = sign(OTS)$. The resulted notation of commitment/decommitment (C) can be seen in the Equation 3.1.7.

$$(C) = (vk, c_1, c_2, \sigma). \tag{3.1.7}$$

- Equiv(pk, a, r, a', c)
  To equiv, it needs to calculate r'

$$r' = \frac{a - a' + H(vk)t_2.r}{H(vk).t2} mod q_8 191 \tag{3.1.8}$$

- Verify(pk, C, a', r')
  To verify the commitment, it verifies these both condition.
  - Signature in signVfy$(vk, c_1||c_2, \sigma)$ is correct
  - Check whether value of c_1 equals to $h_1^{a'} h_2^{hash(vk)r'}$ or not

Chapter 4

Design

Based on the analysis that has been conducted, several things need to be designed to provide the proof that the proposed scheme is more efficient than the other commitment scheme for threshold ECDSA. In this chapter, the input/output that is necessary for the program and the detail of the method will be discussed.

4.1 Input and Output

To implement the commitment scheme, SageMath is chosen as the platform to develop the program because it provides a wide range of crypto library. The software that is developed is a software to analyse the computation time of each commitment scheme listed in the previous chapter when committing messages.
The input and output of the program are as follow:

- Input
  Input for all of the commitment scheme is a Message m, where m is an appended value of u_i and v_i. u_i and v_i itself is a ciphertext from Paillier Cipher which has base modulus n^2 with $n = 2047$ bit.

- Output
  There are two outputs for the commitment scheme as follows:

  1. Commitment c
     Commitment c consist of several values that might be different for each instantiation of a scheme.

  2. Computation time
     Computation time is a time that a scheme need for commiting a message.

4.2 SageMath programs

There are three commitments scheme that will be realised in three SageMath program. The three commitments are the Damgard, MacKenzie, and Galindo. The Damgard commitment is not realised because of the reason that is mentioned in the second chapter. The designs of the SageMath programs which are based on corresponding paper [3] [4] [5] as follows.

4.2.1 Gennaro Commitment

This SageMath program is designed to realise the Gennaro commitment scheme [3]. Constructing large RSA modulus N by selecting p and q large primes which also secure is costly in time. To save time, N is constructed by multiplying p from 8192-bit MODP Group [1] and p from 8192-bit prime[2]. e then is assigned as p prime ffdhe8192[3].

4.2.2 MacKenzie Commitment

This SageMath program is designed to realise the MacKenzie commitment scheme [4]. In the key generation algorithm, generating a p random large secure prime number is costly. To save time, p is assigned as p in prime ffdhe8192 [4] so that the $q = \frac{p-1}{2}$ and $g = 2$.

4.2.3 Galindo Commitment

This SageMath program is designed to realise the Galindo commitment scheme from the third chapter. The scheme is working in two groups of (p_8192, q_8191, g_8191) and (p_2048, q_224, g_224). By using the same reason as the previous scheme, to save time, (p_8192, q_8191, g_8191) is assigned as p and g in 8192-bit MODP Group[5]. For the second group, (p_2048, q_224, g_224) is assigned as in 2048-bit MODP Group with 224-bit Prime Order Subgroup [6]

[1] https://www.ietf.org/rfc/rfc3526.txt
[2] http://www.floatingdoghead.net/bigprimes.html
[3] https://tools.ietf.org/html/draft-ietf-tls-negotiated-ff-dhe-10
[4] https://tools.ietf.org/html/draft-ietf-tls-negotiated-ff-dhe-10
[5] https://www.ietf.org/rfc/rfc3526.txt
[6] https://tools.ietf.org/html/rfc5114

Chapter 5

Implementation and Testing

In this chapter, the software enviroment and the testing result of the commitments is described.

5.1 Implementation

In this section, the detail of the software environment is described.

5.1.1 Hardware Environment

The detail of the hardware that is used to implement and test the commitment scheme is as follows:

1. Notebook : ASUS N550JX
2. Processor: Intel (R) Core(TM) i7-4720HQ CPU @2.60GHz 2.59 GHz
3. Memory : 8192 MB RAM
4. Video Card :NVIDIA GeForce GTX 950M 8078 MB

5.1.2 Software Environment

The Softwares that are used to build to develop the commitment schemes is as follows:

1. Windows 10 Home 64-bit (Host PC)
2. Virtualbox 5.1.2
3. Ubuntu 14.0.4 LTS 64 bit (run as a virtual machine) The hardware setting for this virtual machine is as follows:

- Processor : 4 Core
- Memory : 4096 MB RAM
- Video : 128 MB

4. Sagemath 7.3 (run inside Ubuntu)

5.2 SageMath Program

In this section, implementation of each commitment scheme on SageMath will be discussed. All of the code is implemented by myself according to the mathematical construction for each scheme. Some values for p, q and g are assigned with numbers from external website as previously explained in the Chapter 4 Design. Several libraries that are imported/used such as the **time** to calculate the computation time, **hashlib** for hash function, **AES from Crypto.Cipher** for implementing AES.

5.2.1 Gennaro

Figure 5.1: Key Generation Gennaro Commitment

In Figure 5.1, rsa_kg() shows how to realise the Key Generation algorithm of the Gennaro commitment.

```
# Commit to a element of 1 ... 2^l-1
def commit(a):
    # r = random element from Zn*
    while True:
        r = ZZ.random_element(N);
        if gcd(r,N) == 1: break
    #print "r ", r;

    # A = Commitment = (s^a * r^e)mod N
    A = Mod(power_mod(s,a,N)*power_mod(r,e,N),N);
    return [A,[a,r]];
```

Figure 5.2: Commit Gennaro Commitment

5.2.2 MacKenzie

Figure 5.3: Implementation of MacKenzie Commitment Scheme

In Figure 5.3, it is shown the key generation algorithm which is represented in TCgen() and the commit algorithm is represented in TCcom(). The value of the p and q is following the design on the Chapter 4 design.

5.2.3 Galindo

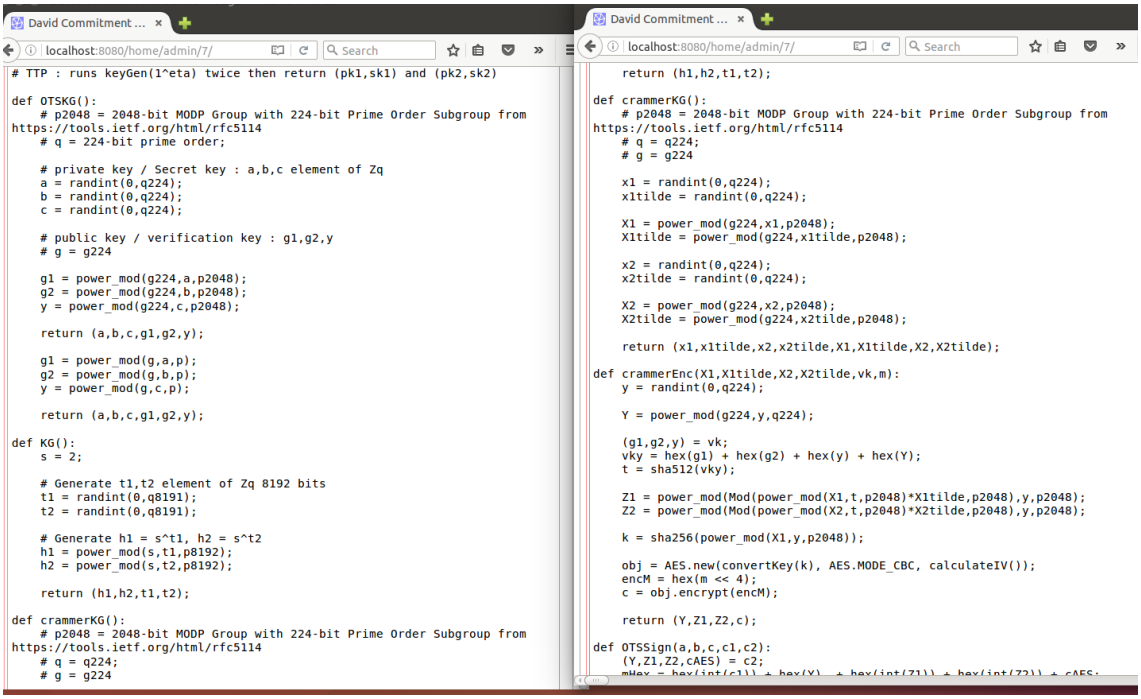

Figure 5.4: Key Generation of Galindo Commitment

Figure 5.4 shows how to implement the Key Generation for One time signature, GGRO8, and Cramer Shoup. The implementation of those three key generation is straight forward. It just need to follow the equation based on the Chapter 3 with the value of p, q and g that is designed in Chapter 4. There is also Cramer-Shoup encryption on that Figure. Before t can hash the value of vk, vk needs to be converted into hex string first. vk itself consist of g1,g2,y. So the hash value of vk is equal to the appended value of the hex string of $g1, g2, y$. On the $encM$, the message is shifted left by 4 bits (padding) so that it can be processed by the block cipher.

5.2.4 Galindo

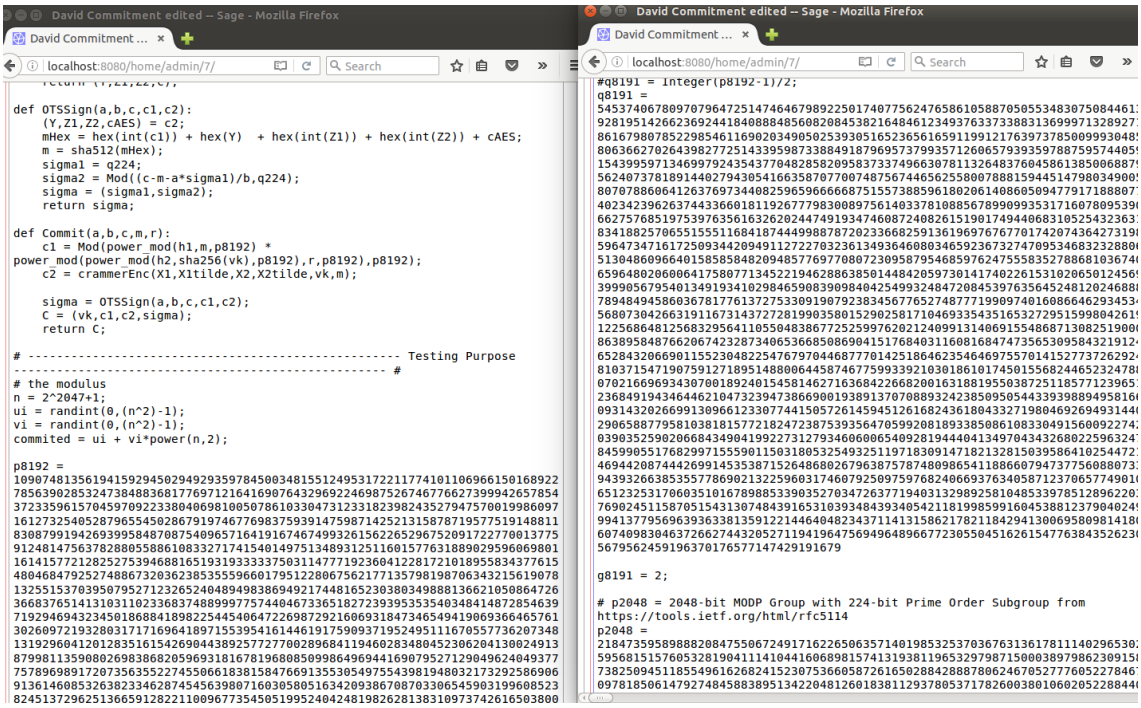

Figure 5.5: One Time Signature and Commit of Galindo Commitment

Figure 5.5 shows the sign implementation of the one-time signature based on DLP. Before processed into the SHA512 to get m, the value of $c1, Y, Z_1, Z_2, c_2$ needs to be converted into a hex string first. The commit function of the Galindo scheme is a compilation of the Cramer-Shoup and the OTS. The rest part of the Figure 5.5.

5.2.5 Galindo

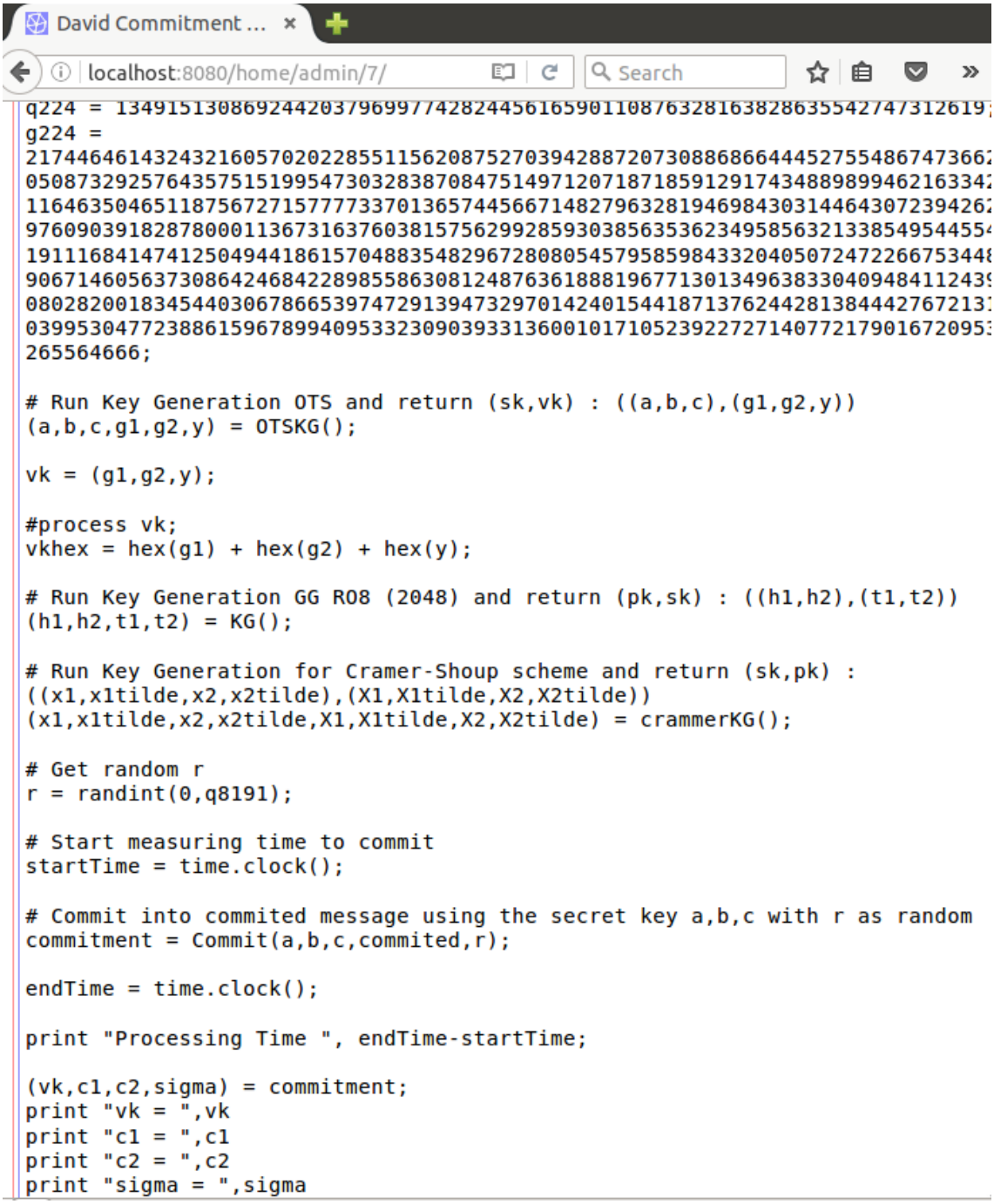

Figure 5.6: Implementation of MacKenzie Commitment Scheme

Some part of Figure 5.5 with Figure 5.6 shows how to instantiate the input message with the two groups of (p_8192, q_8191, g_8191) and (p_2048, q_224, g_224) values following the design in Chapter 4. Figure 5.6 shows the flow of the Galindo scheme after the instatiation of the input message and two group (p, q, g) which is to run the OTS

Key Generation, key generation GGR08, key generation of the Cramer-Shoup then find random value r and start committing and calculating the process time.

5.3 Testing Design

Quantitative testing is chosen for this project. It is conducted by measuring the computation time for each scheme when it is about to commits a message m. The message here is a pair of (u_i, v_i) where each u_i, v_i is a random element from Z_{N^2}. The value of N is equals to $2^{2047} + 1$. m, appended value of u_i v_i, is equals to $(u_i + v_i^2)$. The processing time for each scheme is used to determine which one is more efficient. Less time taken means more efficient of that commitment scheme is. Each scheme will try to commit the same message 30 times.

Chapter 6

Results and evaluation

In this chapter the results of implementation scheme in the previous chapter will be presented. After that the results will be evaluated and commented.

6.1 SageMath program results

In this section, the result/output for each scheme will be shown whether it is successfully returns the computation time and the commitment/decommitment pair or not.

6.1.1 Galindo Scheme

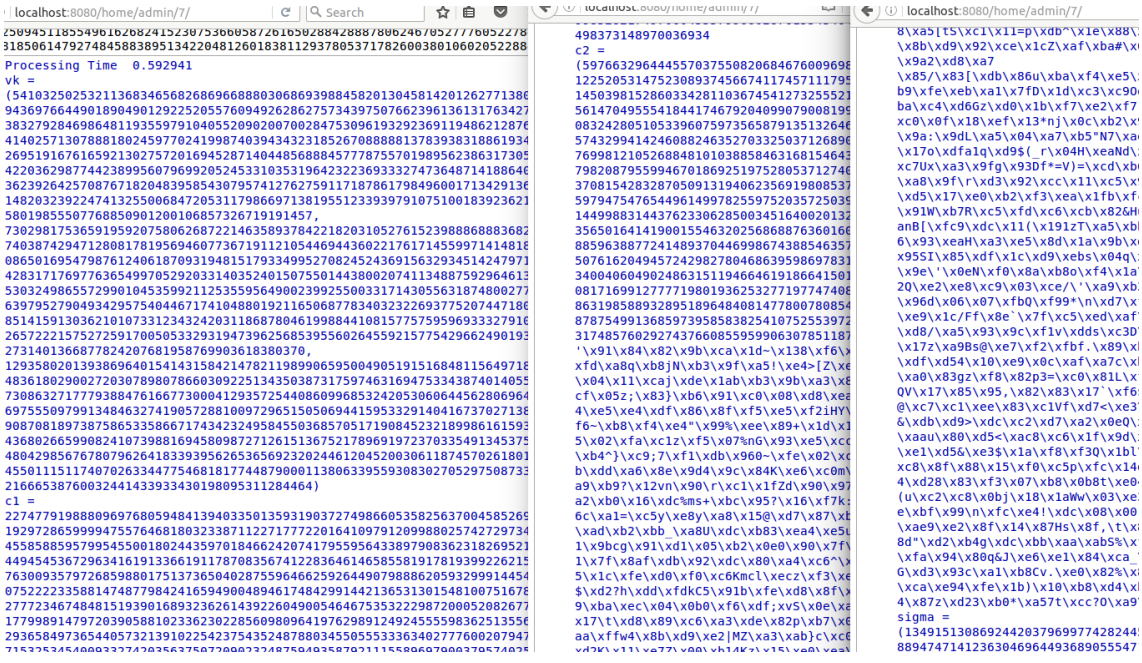

Figure 6.1: Commiting Galindo Scheme Commitment

Figure 6.1 shows that it is successfully to commit message and return the commitment/decommitment pair $((vk, c1, c2, sigma), (m, r_1))$. It took 0.592941 seconds to commit a message using this scheme.

6.1.2 Gennaro Scheme

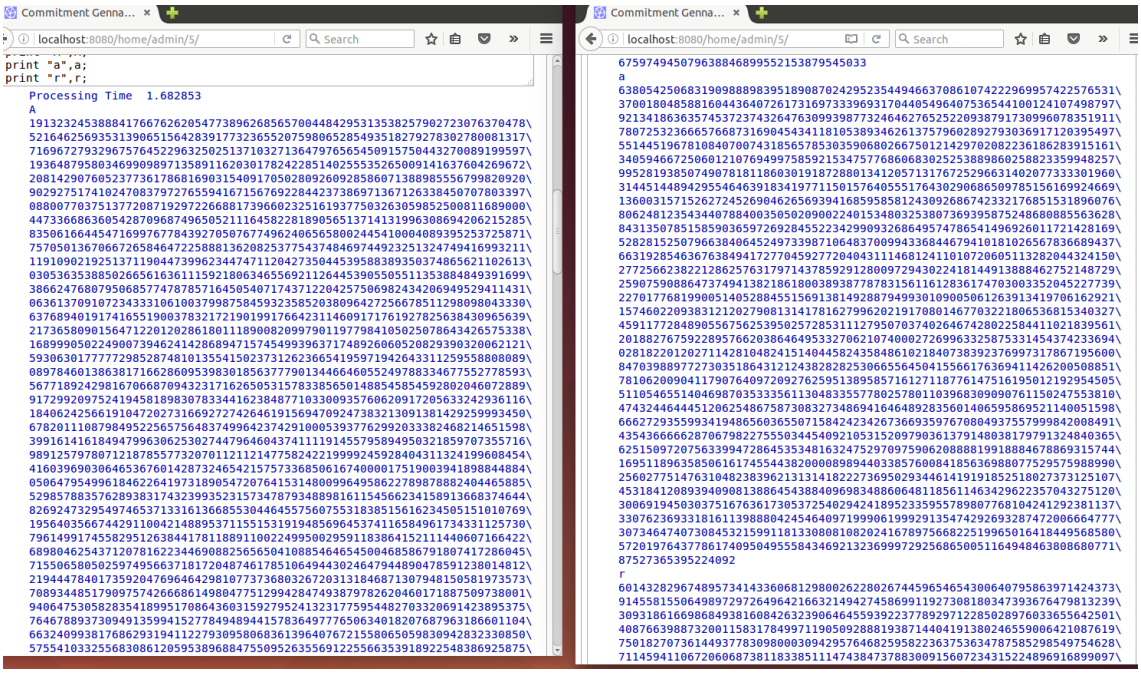

Figure 6.2: Commiting Gennaro Scheme Commitment

Figure 6.2 shows that it is successfully to commit message and return the commitment/decommitment pair $(C, D) = (A, (a, r))$. it took 1.682853 seconds to commit a message using this scheme.

6.1.3 MacKenzie Scheme

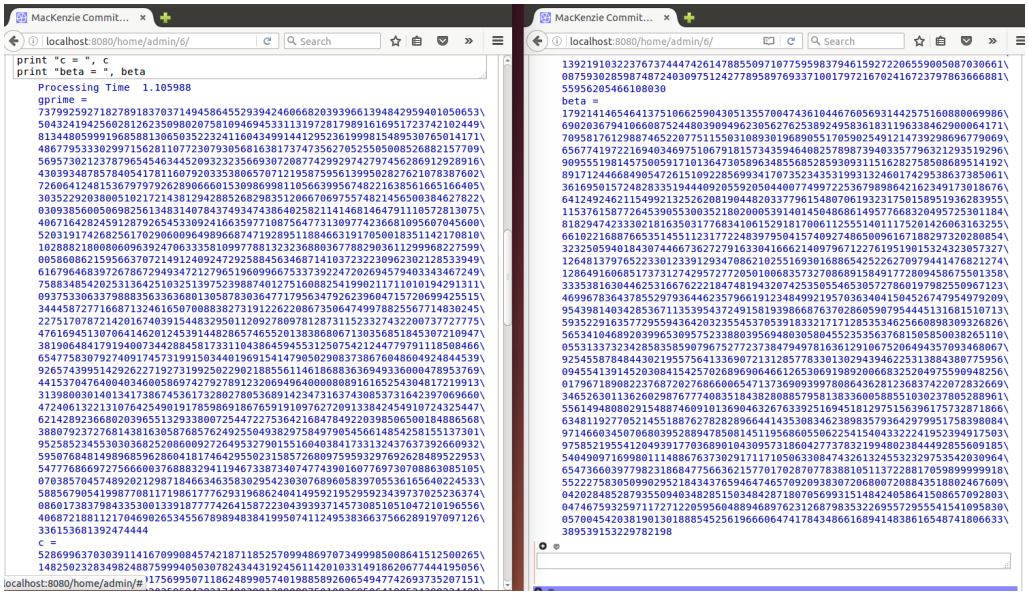

Figure 6.3: Commiting MacKenzie Scheme Commitment

Figure 6.3 shows that it is successfully to commit message and return the commitment/decommitment pair $(C, D) = ((g', c), \beta)$. it took 1.105988 seconds to commit a message using this scheme.

6.2 Testing Result

In this section, the result of qualitative testing designed in the Chapter 5 will be presented. The result can be seen at the Table 6.1. .

Attempt No.	Galindo	Gennaro	MacKenzie
1	0.589088	1.636936	1.11724
2	0.578257	1.628937	1.1362
3	0.59237	1.639142	1.192863
4	0.593226	1.643474	1.099196
5	0.588343	1.607135	1.122539
6	0.594368	1.617173	1.113508
7	0.588432	1.649609	1.1289
8	0.597854	1.644068	1.121914
9	0.598602	1.635983	1.131026
10	0.588397	1.648733	1.120118
11	0.582843	1.634325	1.114878
12	0.590297	1.645597	1.121255
13	0.578146	1.630017	1.116755
14	0.573431	1.648403	1.123419
15	0.58497	1.639231	1.130935
16	0.583826	1.636228	1.118372
17	0.580681	1.637478	1.123152
18	0.591241	1.658114	1.139124
19	0.594737	1.634191	1.137118
20	0.591177	1.621906	1.128035
21	0.592282	1.651259	1.100836
22	0.594414	1.624751	1.113989
23	0.597776	1.608375	1.109527
24	0.587701	1.641879	1.111538
25	0.58151	1.638917	1.138395
26	0.5974	1.632864	1.127042
27	0.594399	1.64017	1.105434
28	0.588066	1.598325	1.106984
29	0.590659	1.633595	1.113115
30	0.602909	1.632979	1.115902

Table 6.1: Computation Times of Committing Message 30 Times

Each row in Table 6.1, represent computation time for each attempt of each scheme in second. It is seen that for every attempt of committing a message, Galindo scheme is faster than the rest of the scheme. 6.2 represent its computation time to

commit message.

6.2.1 Evaluation

Based on the Table 6.1, Table 6.2 that represent the average computation time for commiting a message can be constructed.

Scheme	Time
Galindo	0.589580067
Gennaro	1.6346598
MacKenzie	1.122643633

Table 6.2: Computation Time For Each Scheme To Commit Message

It can be seen that on average (in this testing is 30 attempt), Galindo scheme is faster than the rest of the scheme. The computation time of Gennaro is 2.772583221 times slower whereas MacKenzie is 1.904141094 times slower. Overall the proposed scheme committing computation time is a lot faster by around 50%. So it is proven that the proposed scheme is more efficient.

Chapter 7

Discussion

In this chapter, the achievements the future work will be discussed.

7.1 Achievements

The motivation for this project is fulfilled which is to propose an improvement commitment scheme that is applicable for the threshold DSA/ECDSA. The project has been successfully to realise Gennaro, MacKenzie, and Galindo scheme in a SageMath program as a proof of concept that the proposed scheme is more efficient.

7.2 Future Work

For the further work can try to study on the complete cryptosystem of the threshold DSA/ECDSA then applied the proposed Galindo commitment scheme. By building the whole threshold DSA/ECDSA with different commitment scheme it can be further proven that the proposed scheme is more efficient.

Chapter 8

Conclusion

Threshold DSA/ECDSA is one solution to mitigate malware stealing Bitcoin. The protocols behind the threshold DSA/ECDSA are complex. Commitment is one of that protocol that allows a sender to commmit to a message with information privacy. There are Damgard, Gennaro, MacKenzie commitment that is applicable for the threshold DSA/ECDSA. Galindo scheme is proposed as an improvement for the threshold DSA/ECDSA. By implementing and testing the realisation for each scheme, it is proven that on average, the Galindo scheme is a lot faster by around 50%.

Bibliography

[1] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security," in *International Conference on Applied Cryptography and Network Security*, pp. 156–174, Springer, 2016.

[2] I. Damgard and J. Groth, "Non-interactive and reusable non-malleable commitment schemes," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pp. 426–437, ACM, 2003.

[3] R. Gennaro, "Multi-trapdoor commitments and their applications to proofs of knowledge secure under concurrent man-in-the-middle attacks," in *Annual International Cryptology Conference*, pp. 220–236, Springer, 2004.

[4] P. MacKenzie and K. Yang, "On simulation-sound trapdoor commitments," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 382–400, Springer, 2004.

[5] D. Galindo, F. D. Garcia, and P. Van Rossum, "Computational soundness of non-malleable commitments," in *International Conference on Information Security Practice and Experience*, pp. 361–376, Springer, 2008.

[6] D. Cash, E. Kiltz, and V. Shoup, "The twin diffie-hellman problem and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 127–145, Springer, 2008.

[7] M. Abe, Y. Cui, H. Imai, and E. Kiltz, "Efficient hybrid encryption from id-based encryption," *Designs, Codes and Cryptography*, vol. 54, no. 3, pp. 205–240, 2010.

[8] G. Gutoski and D. Stebila, "Hierarchical deterministic bitcoin wallets that tolerate key leakage," in *International Conference on Financial Cryptography and Data Security*, pp. 497–504, Springer, 2015.

[9] E. Biham, "Secret sharing." http://www.cs.haifa.ac.il/ orrd/IntroToCrypto/Spring11/Lecture1 .pdf, June 2011.

[10] H. C. Van Tilborg, *Fundamentals of cryptology: a professional reference and interactive tutorial*, vol. 528. Springer, 1999.

Appendix A

SVN project repository

The address of our SVN project repository is
https://codex.cs.bham.ac.uk/svn/projects/2015/mnw529/

A.1 Contents of the SVN project repository

The SVN contains three implementations of the commitments scheme namely Galindo Commitment.sws, Gennaro Commitment.sws, MacKenzie Commitment.sws in a form of SageMath program. The program will produce the commitment of a message and its computation time corresponding to each scheme.

A.2 How to run our software

In order to run the SageMath program, there are steps that can be followed.

- Install SageMath (Installation steps for Windows [1] or Linux [2]))
- Run the Sagemath application in Notebook mode.
- Upload all of the commitment file from the SVN to the SageMath.
- Open the commitment scheme that is going to be tested its computation time.
- Choose "evaluate all" to run all of the command in the code and to see the resulted commitment and computation time.

.

[1] http://www.sagemath.org/download-windows.html
[2] http://www.sagemath.org/download-linux.html